Volatile具有synchronized的可见性特征，但不具备原子性，不能用于计数标志。

1. **系统的内存模型**

计算机通过cpu执行程序中的指令，指令涉及的变量都存储在主存中，所以需要涉及内存的读取和写入。但是cpu的执行速度远高于内存读取和写入速度，所以就有了cpu的高速缓存。当程序运行时会从主存中拷贝一份数据到cpu的高速缓存中，cpu执行指令就从高速缓存中读取和写入，当程序执行完后再将结果重新写入主存中。

1. **并发的三个概念**：

1. 原子性：一个操作或者多个操作要么全部执行并且执行过程中不会被打断，要么全部不执行。比如：银行转账

2. 可见性：当多个线程访问一个变量，其中一个线程修改了这个变量的值，其他线程也可以立刻看到变化。

3. 有序性：程序的执行顺序按照代码的先后顺序执行

1. **指令重排序**

cpu为了提高运行效率，他会对输入代码进行优化，他不保证程序中的指令的执行顺序同代码的顺序一致，但是他保证程序的最终执行结果同代码顺序执行的结果一致。但这只对单线程，多线程不保证。

1. **Java的内存模型**

java的内存模型规定所有的内存模型都放在主存中，每个线程都有自己的工作内存（类似前面的高速缓存），线程对变量的所有操作必须在工作内存中，不能直接对主存操作，而且线程不能访问其他线程的工作内存。

Java对原子性，可见性，有序性的保障：

1.原子性：只保障简单的读取、赋值（而且必须是将数字赋值给某个变量，变量之间的相互赋值不是原子操作）才是原子操作。

synchronized和lock可保证原子性，因为他们规定代码段任何时刻只有一个线程可以访问，自然不会存在原子性问题。

2.可见性：java提供volatile保证可见性。

另外，synchronized和lock也能保证可见性。因为synchronized和lock保障任何时刻只有一个线程可以获得锁并执行代码，而且在锁释放之前就会将结果返回到主存上，可以保证可见性。

3.有序性:java内存模型运行cpu指令重排序，所以在单线程没问题，但会影响多线程的正确性。

但是java 内存模型有先天优势，也就是happens-before原则（先行发生原则）：

* 程序次序规则：一个线程内，按照代码顺序，书写在前面的操作先行发生于书写在后面的操作
* 锁定规则：一个unLock操作先行发生于后面对同一个锁额lock操作
* volatile变量规则：对一个变量的写操作先行发生于后面对这个变量的读操作
* 传递规则：如果操作A先行发生于操作B，而操作B又先行发生于操作C，则可以得出操作A先行发生于操作C
* 线程启动规则：Thread对象的start()方法先行发生于此线程的每个一个动作
* 线程中断规则：对线程interrupt()方法的调用先行发生于被中断线程的代码检测到中断事件的发生
* 线程终结规则：线程中所有的操作都先行发生于线程的终止检测，我们可以通过Thread.join()方法结束、Thread.isAlive()的返回值手段检测到线程已经终止执行
* 对象终结规则：一个对象的初始化完成先行发生于他的finalize()方法的开始

Java还可以通过volatile和synchronized，lock来保证有序性。

1. **Volatile**

1.Volatile保证所描述的变量不执行执行重排，但不保证原子性，比如x++。

2.volatile适合的场景：

条件：

1）对变量的写操作不依赖于当前值

2）该变量没有包含在具有其他变量的不变式中

场景：

3）常作为标记状态

4）. double check

3.原理

生成汇编编码会多一个lock指令。

1）它确保指令重排序时不会把其后面的指令排到内存屏障之前的位置，也不会把前面的指令排到内存屏障的后面；即在执行到内存屏障这句指令时，在它前面的操作已经全部完成；

2）它会强制将对缓存的修改操作立即写入主存；

3）如果是写操作，它会导致其他CPU中对应的缓存行无效。